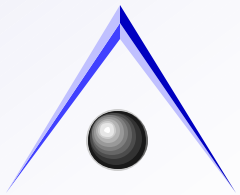
Mapping Algorithms to the Network Topology in a Portable Manner

Dave Turner

In collaboration with:

Bin Tong, Ashraf Hamad, Masha Sosonkina

Funded by the Mathematical, Information, and Computational Sciences (MICS) division of the Department of Energy Office of Science



Outline

- The importance of mapping to the network topology
  - The topology of the IBM SP at NERSC
  - On-node and off-node performance measurements
  - The Ames Laboratory Classical Molecular Dynamics code
  - Performance on the IBM SP with and without mapping
- 2D and 3D decomposition of applications
- Taking advantage of the network topology
  - The goals of the NodeMap project
  - Automatically mapping 2D decompositions onto the network
  - Examples of NodeMap modules

The IBM SP at NERSC

- 416 16-way SMP nodes connected by an SP switch
- **380 IBM Nighthawk compute nodes** → 6080 compute processors
  - **16-way SMP node**
  - 375 MHz Power3+ processors
    - 4 Flops / cycle → peak of 1500 MFlops/processor
    - ALCMD gets around 150 MFlops/processor
  - 16 GB RAM / node (some 32 & 64 GB nodes)
    - Limited to 2 GB / process
  - AIX with MPI or OpenMP
- **IBM Colony switch** connecting the SMP nodes
  - **2 network adaptors per node**
  - MPI

http://hpcf.nersc.gov/computers/SP/

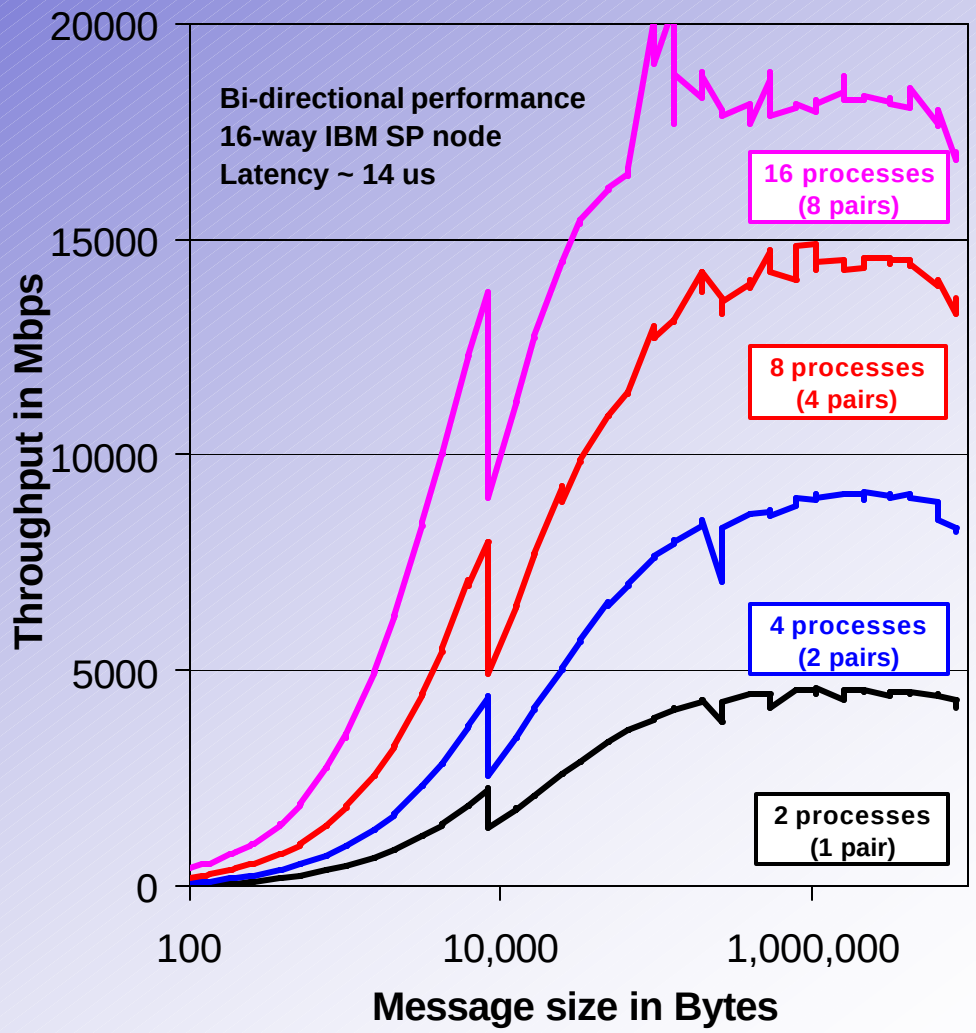
SMP message-passing performance on an IBM SP node

The aggregate bi-directional bandwidth is **4500 Mbps** between **one pair** of processes on the same SMP node with a latency of **14 us**.

The bandwidth scales ideally for **two pairs** communicating simultaneously.

Efficiency drops **80%** when **4 pairs** are communicating, saturating the main memory bandwidth on the node.

Communication bound codes will suffer when run on all **16 processors** due to this saturation.



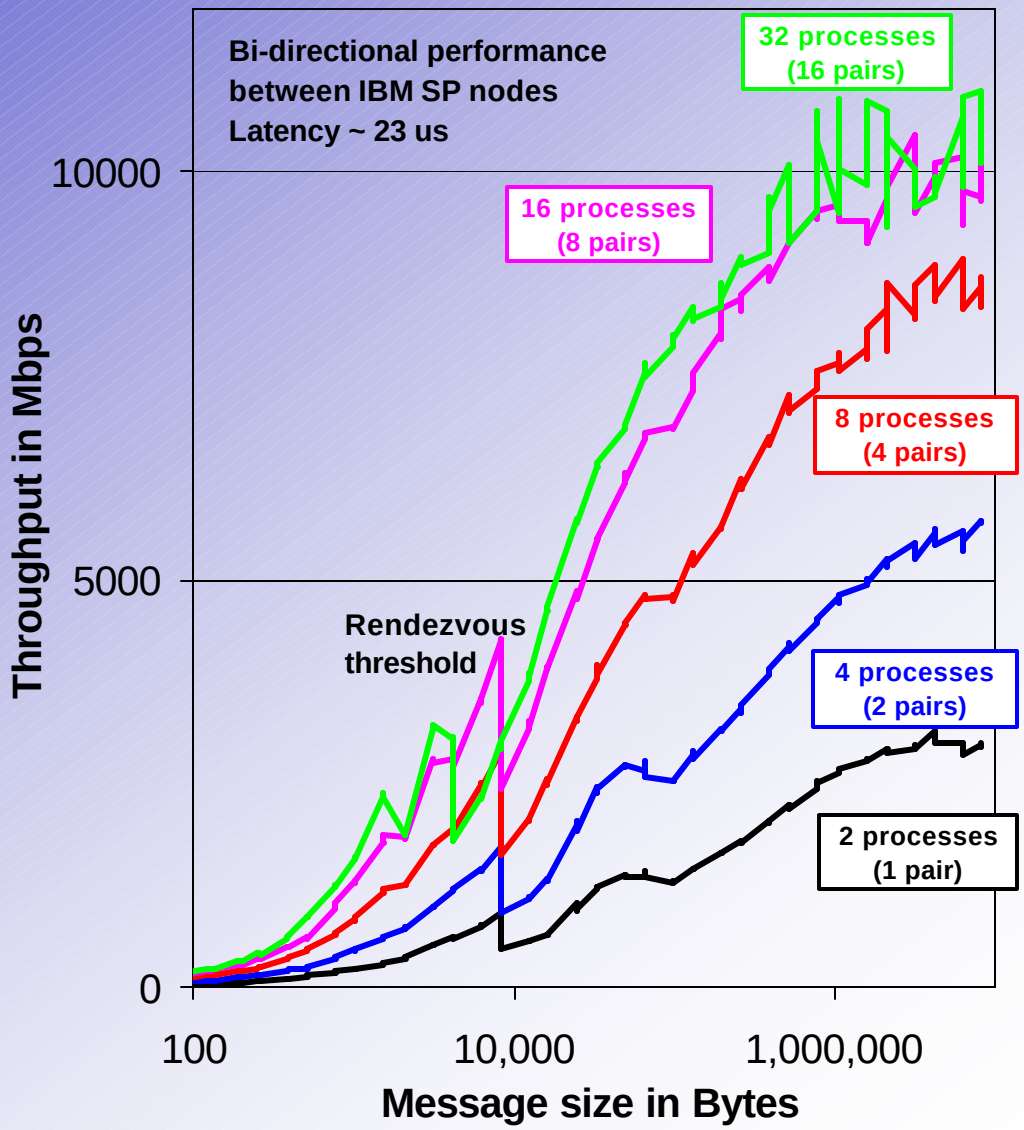
Message-passing performance between IBM SP nodes

The aggregate bi-directional bandwidth is **3 Gbps** between **one pair** of processes between nodes with a latency of **23 us**.

The bandwidth scales ideally for **two pairs** communicating simultaneously, which makes sense given that there are **two network adapter cards per node**.

Efficiency drops **70%** when **4 pairs** are communicating, just about saturating the off-node communication bandwidth.

Communication bound codes will suffer when run on **more than 4 processors** due to this saturation.



Ames Lab Classical Molecular Dynamics code

Embedded atom method, Leonard Jones, Tersoff potentials

Uses cubic spline functions for optimal performance

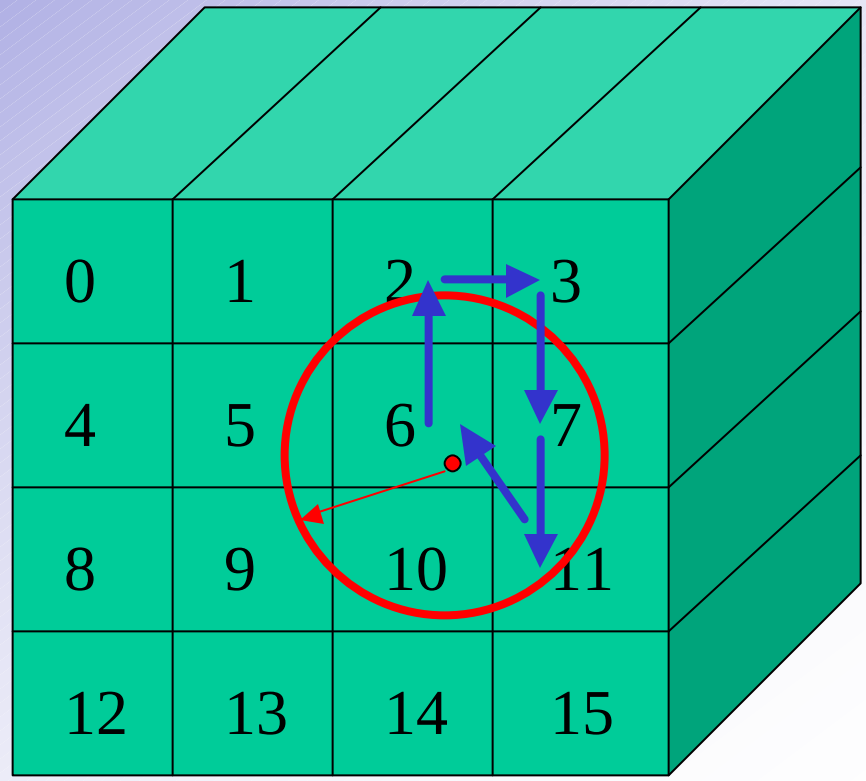
Local interactions only, typically 5-6 A → 50-60 neighbors per atom

2D decomposition of the 3D simulation space

Map neighboring regions to neighboring nodes to localize communications.

Shift coordinates and accumulators to all nodes above and to the right to calculate all pair interactions within the cutoff range.

Large systems require just 5 communications, while systems spread across more nodes may involve many more passes in a serpentine fashion around half the interaction range.



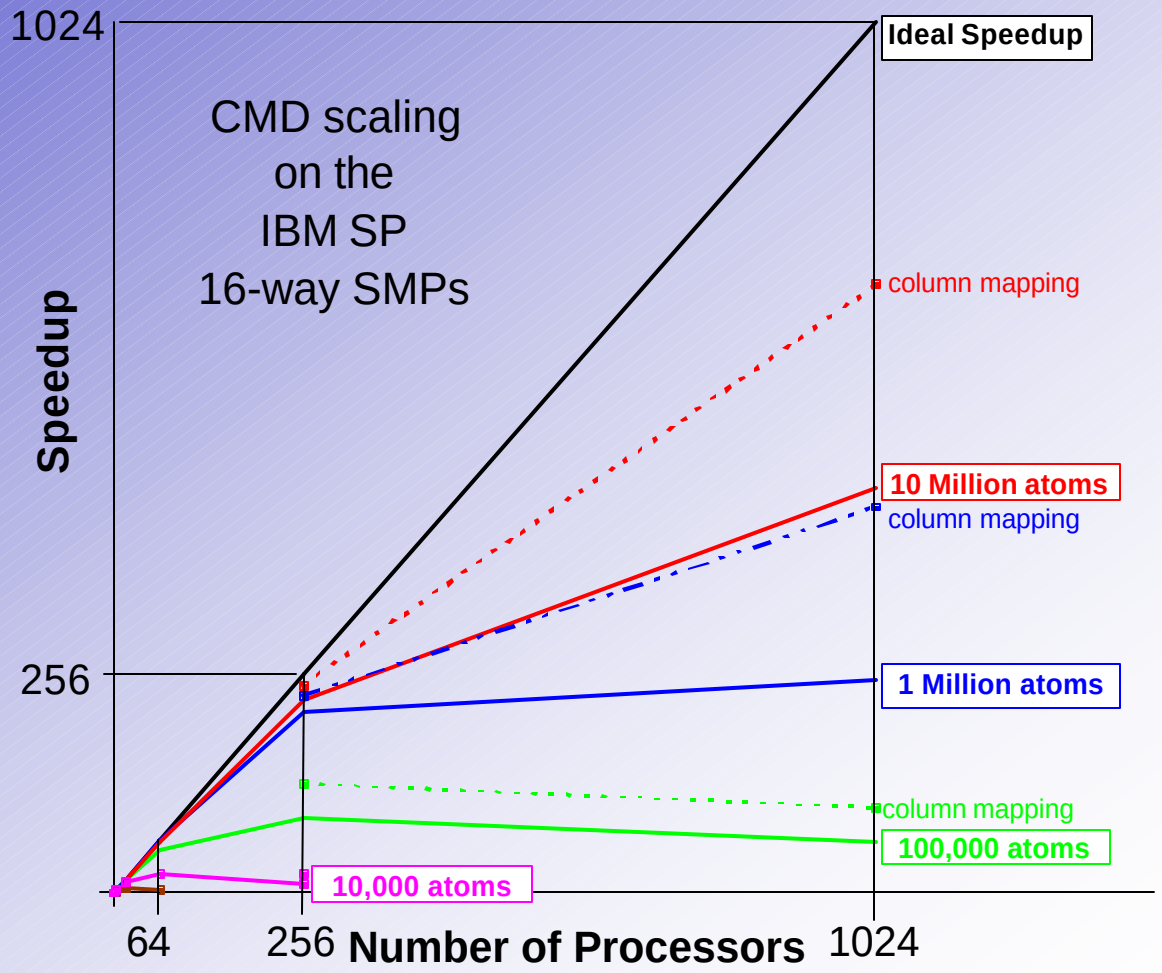
ALCMD scaling on the IBM SP

Proper mapping of the columns to SMP nodes helps greatly.

Parallel efficiency goes from **50% to 70%** for **10,000,000 atoms** on 1024 processors.

Scaling beyond 1024 processors will be difficult.

On-node and off-node communications are saturated even at 1024 processors (16 x 16-way SMPs)



2D decomposition of algorithms

- Many codes can be naturally decomposed onto a 2D mesh.
  - Atomistic codes: Classical and Order-N Tight-Binding
  - Many matrix operations
  - Finite difference and finite element codes
  - Grid and multi-grid approaches
- 2D decompositions map well to most network topologies.
  - 2D and 3D meshes and toruses, hypercubes, trees, fat trees
  - Direct connections to nearest neighbor nodes prevents contention
- Writing algorithms using a 2D decomposition can provide the initial step to taking advantage of the network topology.

NodeMap

- Most applications do not take advantage of the network topology

  → There are many different topologies, which can even vary at run-time

  → Schedulers usually do not allow requests for a given topology

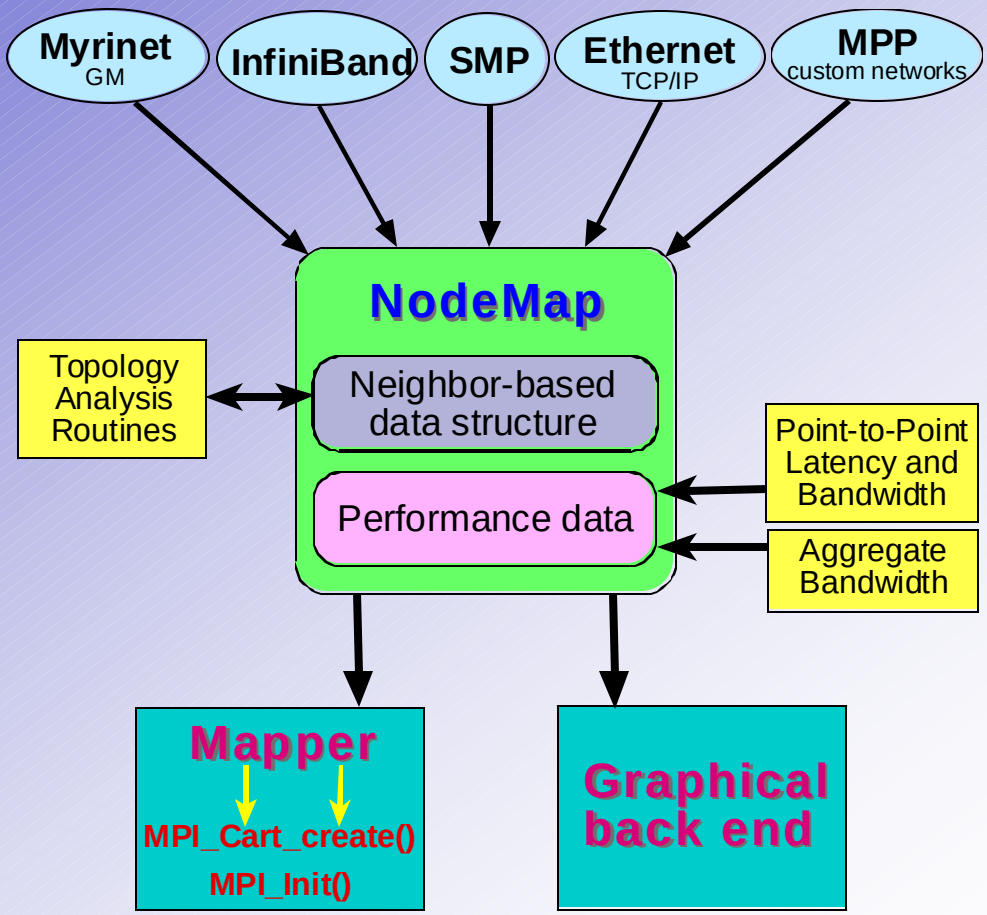
  → no portable method for mapping to the topology

  ➔ loss of performance and scalability

- **NodeMap** will automatically determine the network topology at run-time and pass the information to the message-passing library.

NodeMap

- Network modules identify the topology using a variety of discovery techniques.
- Hosts and switches are stored from nearest neighbors out.
- Topology analysis routines identify regular topologies.
- Latency, maximum and aggregate bandwidth tests provide more accurate performance measurements.
- The best mapping is provided through the **MPI_Init** or **MPI_Cart_create** functions.



NodeMap network modules

- Run-time use of NodeMap means it must operate very quickly (seconds, not minutes).
  - Use brute force ping-pong measurements as a last resort.
- Reduce the complexity of the problem at each stage.
  - Always identify SMP processes first using ***gethostname***.
  - Identify each node's nearest neighbors, then work outwards.
- Store the neighbor and switch information in the manner it is discovered.
  - Store local neighbors, then 2nd neighbors, etc.
- This data structure based on discovery makes it easy to write the topology analysis routines.
- The network type or types should be known from the MPI configuration.
  - This identifies which network modules to run.
  - The MPI configuration provides paths to the native software libraries.

Myrinet – static.map file

- Parse the gm/sbin/static.map file
- Each host has an entry

  -- Connected to what switch?
- Each switch has an entry

  -- Lists all hosts connected

  -- Lists all switches connected
- Internal switches have no hosts
- Determine the complete topology
- Determine our topology

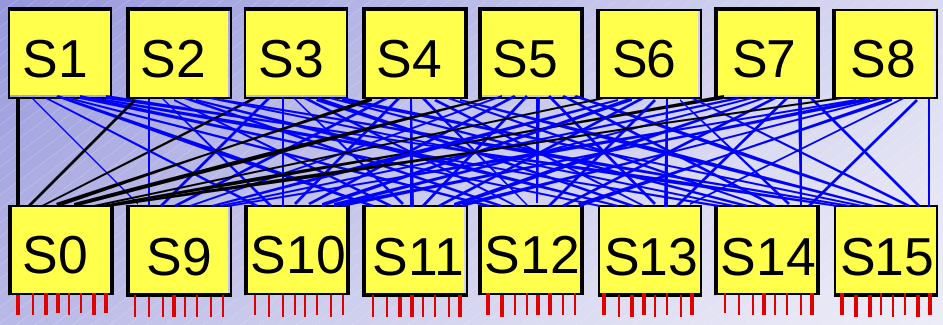
```
h - "m22"
1
0 s - "s0" 12
number 0
address
0060dd7fb1f9
gmId 78
hostType 0

h - "m27"
1
0 s - "s14" 14
number 0
address
0060dd7fb0e8
gmId 62
hostType 0
```

```
s - "s0"
15
0 s - "s1" 14
1 s - "s2" 14
2 s - "s3" 14
3 s - "s4" 14
4 s - "s5" 14
5 s - "s6" 14
6 s - "s7" 14
7 s - "s8" 14
9 h - "a18" 0
10 h - "a19" 0
11 h - "4pack" 0
12 h - "m22" 0
13 h - "m19" 0
14 h - "m18" 0
15 h - "m20" 0

s - "s1"
8
0 s - "s9" 0
1 s - "s10" 0
6 s - "s11" 0
11 s - "s12" 0
12 s - "s13" 0
13 s - "s14" 0
14 s - "s0" 0
15 s - "s15" 0
```

Fat Tree



Myrinet module for NodeMap

- Probe using ***gm_board_info***.
- Use header info to ID board.
- Verify clock rate with ***dmesg***
- Provides exact routing

  - Not really needed
- Measure the latency and bandwidth
- Provide the best mapping

```
4pack>  gm_board_info
  lanai_cpu_version = 0x0900 (LANai9.0)
  max_lanai_speed   = 134 MHz
    (should be labeled: "M3M-PCI64B-2-59521")

gmID MAC Address           gmName Route
---- ----------------- ---------- -------------
   2 00:60:dd:7f:b1:c0        m26 ba bd 88
   3 00:60:dd:7f:b0:e0        m18 83
   4 00:60:dd:7f:b1:f6        m29 ba b3 89
  55 00:60:dd:7f:ac:b2        m25 ba b2 88
  56 00:60:dd:7f:b0:ec        m24 ba bf 86
  58 00:60:dd:7f:b1:06        m20 84
  59 00:60:dd:7f:b0:e3        m19 82
  61 00:60:dd:7f:b1:bd        m28 ba be 86
  62 00:60:dd:7f:b0:e8        m27 ba bf 89
  67 00:60:dd:7f:ac:a8        m23 ba be 89
  77 00:60:dd:7f:ac:b0      4pack 80 (this node)
  78 00:60:dd:7f:b1:f9        m22 ba be 88
  80 00:60:dd:7f:b0:ed        m32 ba b3 88
  93 00:60:dd:7f:b0:e1        m31 ba be 87
```

InfiniBand module for NodeMap

- Probe the subnet manager (***minism*** or other)
- Identify my LID
- Exchange LIDs
- Parse and store the links, switches, and other HCAs
- This is all that is needed to determine the topology

```
opteron1:~> minism InfiniHost0
 minism>d

New Discovered Node
 New Node - Type:CA NumPorts:02 LID:0003
New Discovered Node
 New Link 4x FromLID:0003 FromPort:01 ToLID:0004 ToPort:08

 New Node - Type:Sw NumPorts:08 LID:0004
   No  Link 1x FromLID:0004 FromPort:01
   No  Link 1x FromLID:0004 FromPort:02
   No  Link 1x FromLID:0004 FromPort:03
   No  Link 1x FromLID:0004 FromPort:04
New Discovered Node
  New Link 4x FromLID:0004 FromPort:05 ToLID:0002 ToPort:06
  New Link 4x FromLID:0004 FromPort:06 ToLID:0002 ToPort:05
  New Link 4x FromLID:0004 FromPort:07 ToLID:0009 ToPort:01
  New Link 4x FromLID:0004 FromPort:08 ToLID:0003 ToPort:01

New Discovered Node
 New Node - Type:CA NumPorts:02 LID:0009
  New Link 4x FromLID:0009 FromPort:01 ToLID:0004 ToPort:07
```

IP module for **NodeMap**

- IP interface can be to many types of network hardware.
  - + Ethernet, ATM, IP over Myrinet, IP over InfiniBand, etc.
- How to determine what network cards are present?
  - + ***ifconfig*** provides a list of active interfaces
    - + Does tell what type of network
    - ~ May tell what speed for Ethernet
  - + ***lspci, hinv*** provide a description of what is plugged into the PCI slots
    - ~ Sometimes helpful, but may require a database
- Can measuring latency identify the number of switches in between?
  - - This may require many point-to-point measurements
  - - OS, driver, and NIC can affect the latency themselves
  - + It may identify which nodes are on a local switch
  - ? Can simultaneous measurements be done to make this efficient?
- Use aggregate measurements to probe higher level switches?

MPP modules for **NodeMap**

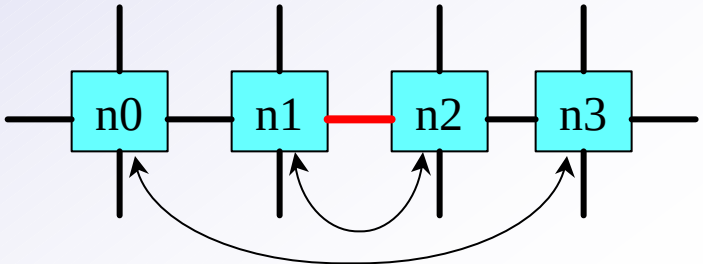
- Use ***uname*** or compiler definitions to identify the MPP type
  - -- **_CRAYT3E** is defined for the Cray T3E
  - -- **_AIX** for IBM SP (anything better?)
  - → These identify the global network topology
- Use ***gethostname*** to identify SMP processes
  - →This reduces the complexity of the problem
- Use vendor functions if available (***getphysnode*** on the Cray T3E)
- Do we need a module for each new type of MPP???

A Generic MPI Module

How quickly can a brute force analysis be done?

Run NodeMap once to generate a static map file of the topology?

- Identify SMP processes first using ***gethostname***
- Gather information for a single host
  - Measure the latency and maximum throughput to one or more nodes
- Probe using overlapping pair-wise bandwidth measurements
  - Try to measure the congestion factor
  - Increase the number of pairs until the performance improves ➔ X-dimension
  - Repeat in the Y-dimension
  - Use 2D global shifts for several regular arrangements
- Measure the bi-sectional bandwidth ➔ Can identify fat trees
- Additional tricks needed!!!



Topological Analysis Routines

- Initially concentrate only on regular arrangements
  - + N-dimensional mesh/torus, trees, SMP nodes
- Identify the topology from the host/switch information gathered
  - + How many unique 1st, 2nd, 3rd, … neighbors ➔ mesh/torus
  - + Determine whether a tree is fully connected
- Eventually handle more irregular arrangements of nodes.
  - + Identify the general type of network
    - - May have an irregular arrangement of nodes on a mesh/torus
  - + Identify which nodes are irregular

Performance Measurements

- Performance may be limited by many factors
  - + Measure latency and max bandwidth for each network layer
  - + Measure aggregate performance across switches or a given link
- Performance data can help determine the topology
  - + A tree with fewer internal switches may still be a fat tree
- Feed the performance data to the Mapper along with topological data

The Mapper

- NodeMap will initially be run from **MPI_Cart_create(…, reorder=1)**
  - + User must determine which direction requires the optimal passing
- The Mapper will take the topology and performance data and provide the best mapping of the desired 2D (or eventually 3D or tree) algorithm.
  - + Concentrate on regular arrangements first
  - + Use Gray codes for 2D onto N-dimensional topologies to guarantee nearest neighbors are directly connect
- NodeMap may also be run from **MPI_Init()**
  - + Provide optimal mappings for global operations (mainly binary trees)

Conclusions

- Codes must be mapped to the network topology to scale well
- Many codes can use a 2D decomposition
  - + 2D algorithms can be mapped ideally to most network topologies
- NodeMap will provide automatic mapping to the topology
  - ➔Portable means of taking advantage of the network topology

Questions

- How well will NodeMap handle irregular networks?
  - Will it be difficult to provide a reasonable mapping?
- Can a generic MPI module effectively discover a topology?
  - If so, how quickly can this be done?
  - Will NodeMap need to generate static map files ahead of time?

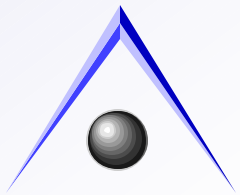
Contact information

Dave Turner - turner@ameslab.gov

http://www.scl.ameslab.gov/Projects/NodeMap/

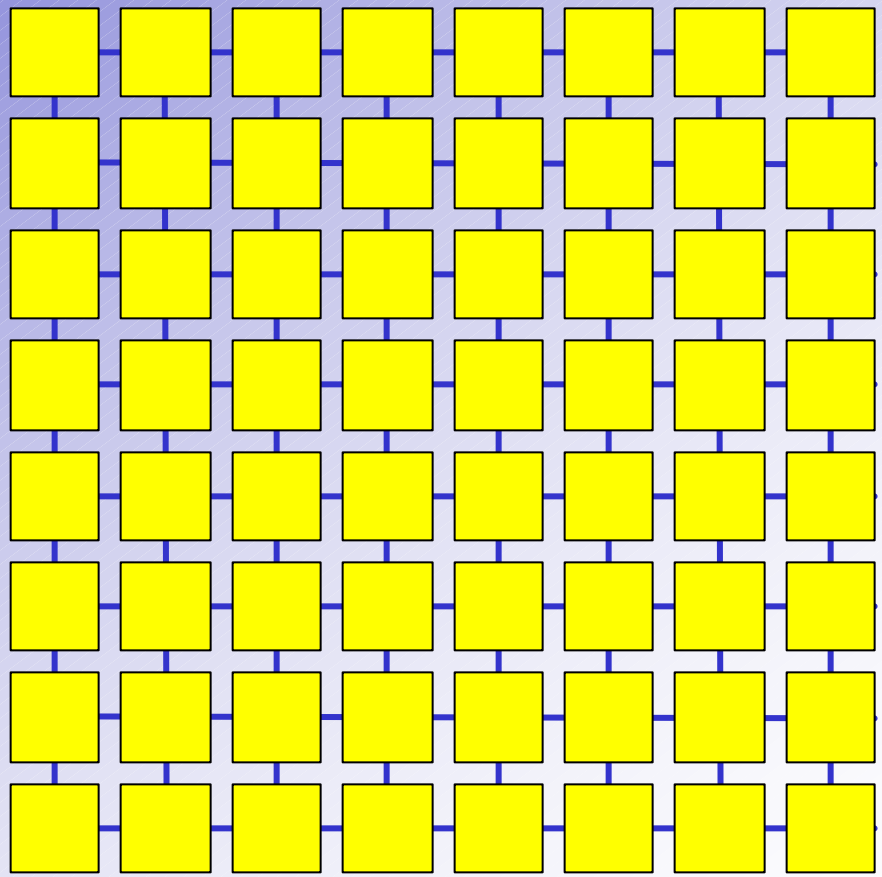
http://www.scl.ameslab.gov/Projects/NetPIPE/

http://cmp.ameslab.gov/cmp/CMP_Theory/cmd/cmd.html



SCI cluster

- 64 dual-Athlon MP1900+ nodes
  - 1.6 GHz
  - 2 GB RAM
- Salable Coherent Interface (SCI)
  - 8x8 grid
  - Directional rings → wrapped at the ends.
  - Very fast line rate to avoid congestion.
- Programming environment
  - Scali MPI (ScaMPI)
  - Intel and Gnu compilers
  - PBS



SCI performance

InfiniBand can deliver **4500 - 6500 Mbps** at a **7.5 us** latency.

Atoll delivers **1890 Mbps** with a **4.7 us** latency.

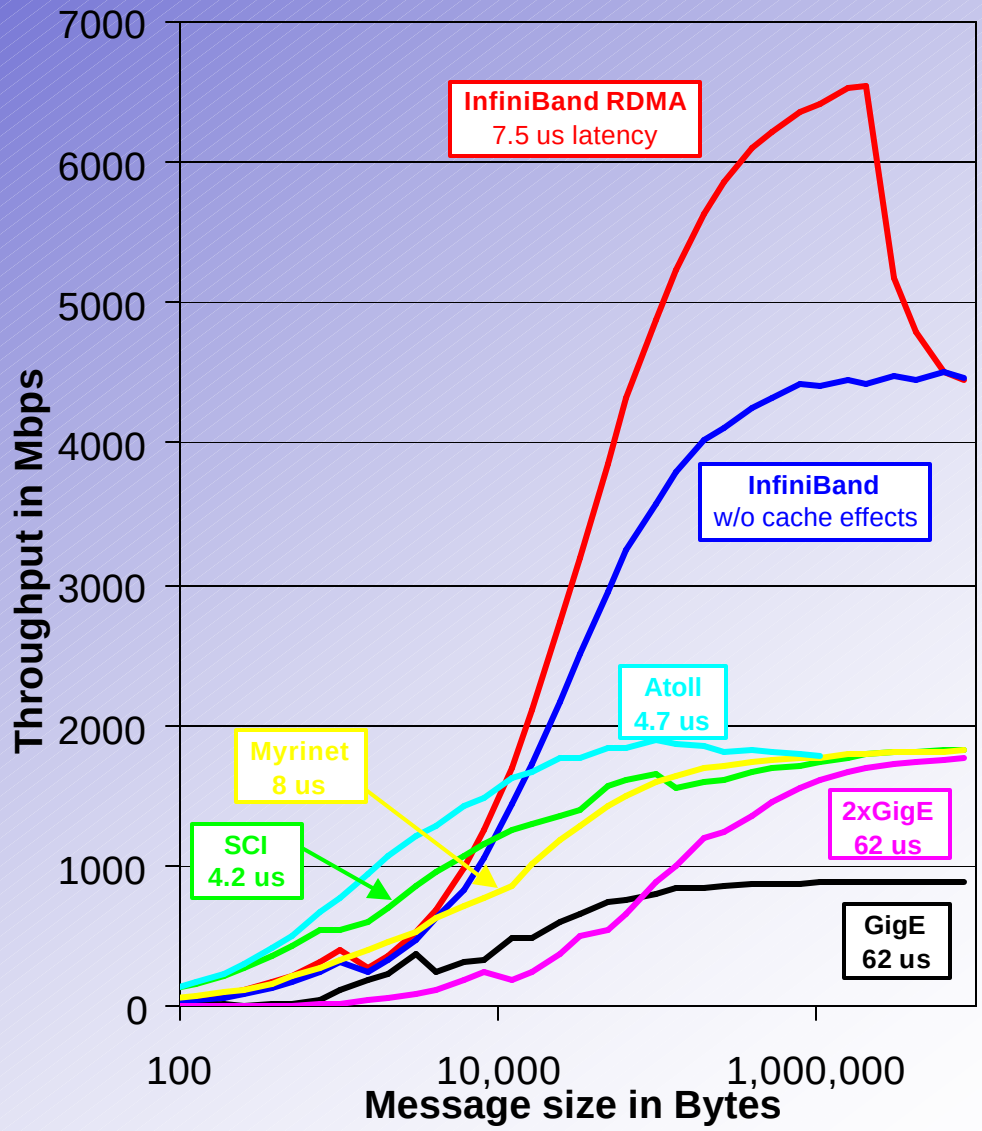
SCI delivers **1840 Mbps** with only a **4.2 us** latency.

Myrinet performance reaches **1820 Mbps** with an **8 us** latency.

Channel-bonded GigE offers **1800 Mbps** for very large messages.

Gigabit Ethernet delivers **900 Mbps** with a **25-62 us** latency.

10 GigE only delivers **2 Gbps** with a **75 us** latency.



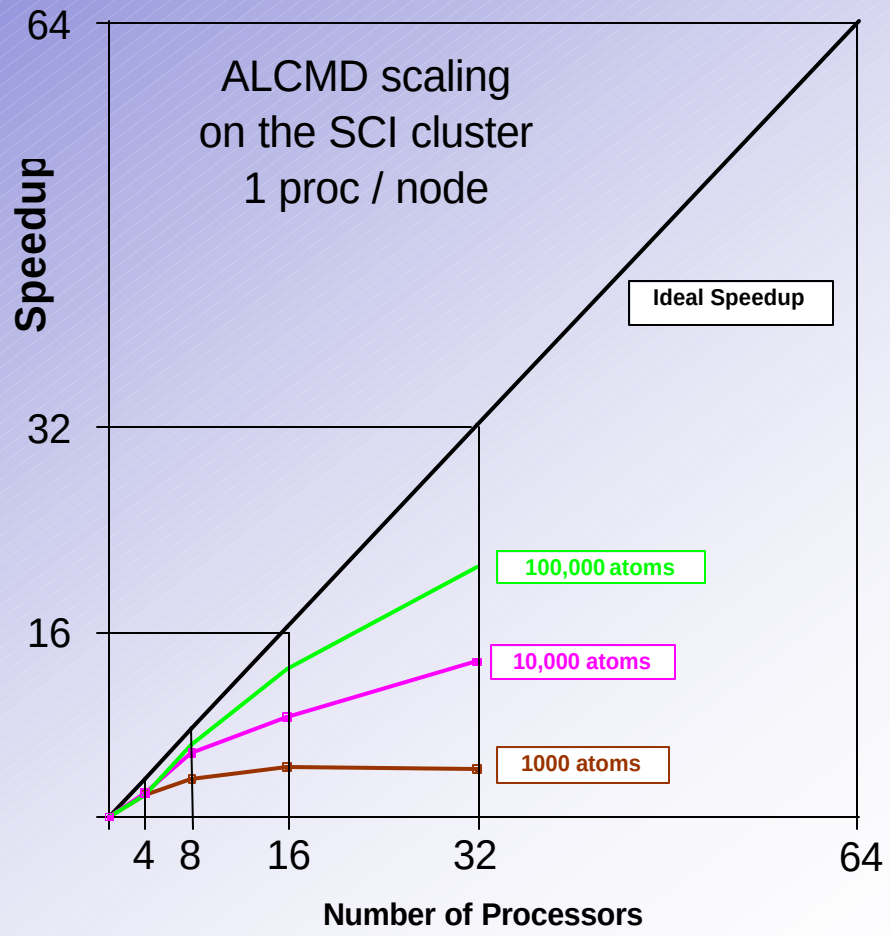
ALCMD scaling on the SCI cluster

Scaling is reasonable as long as the code is mapped to the 2D mesh.

Larger systems will scale even better.

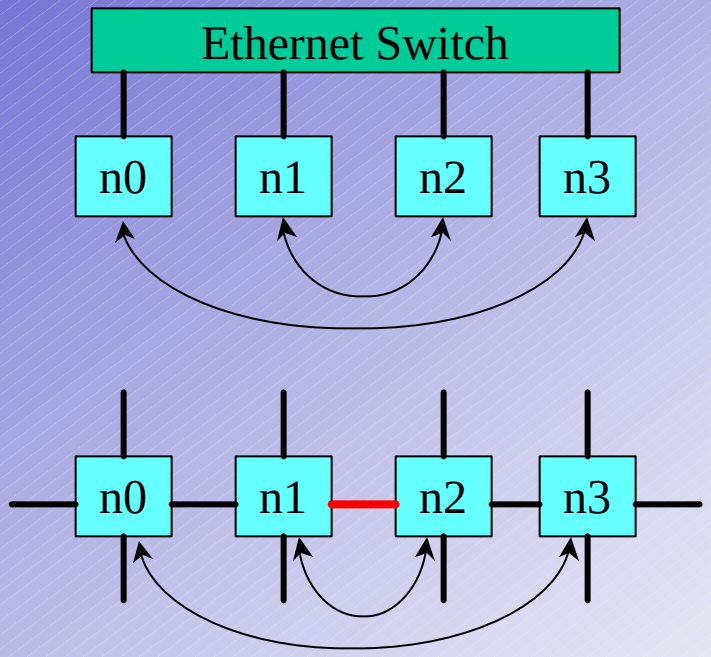
There will be some loss going to 2 processors per node.

Have not tested directional nature of the communication links.



Current projects

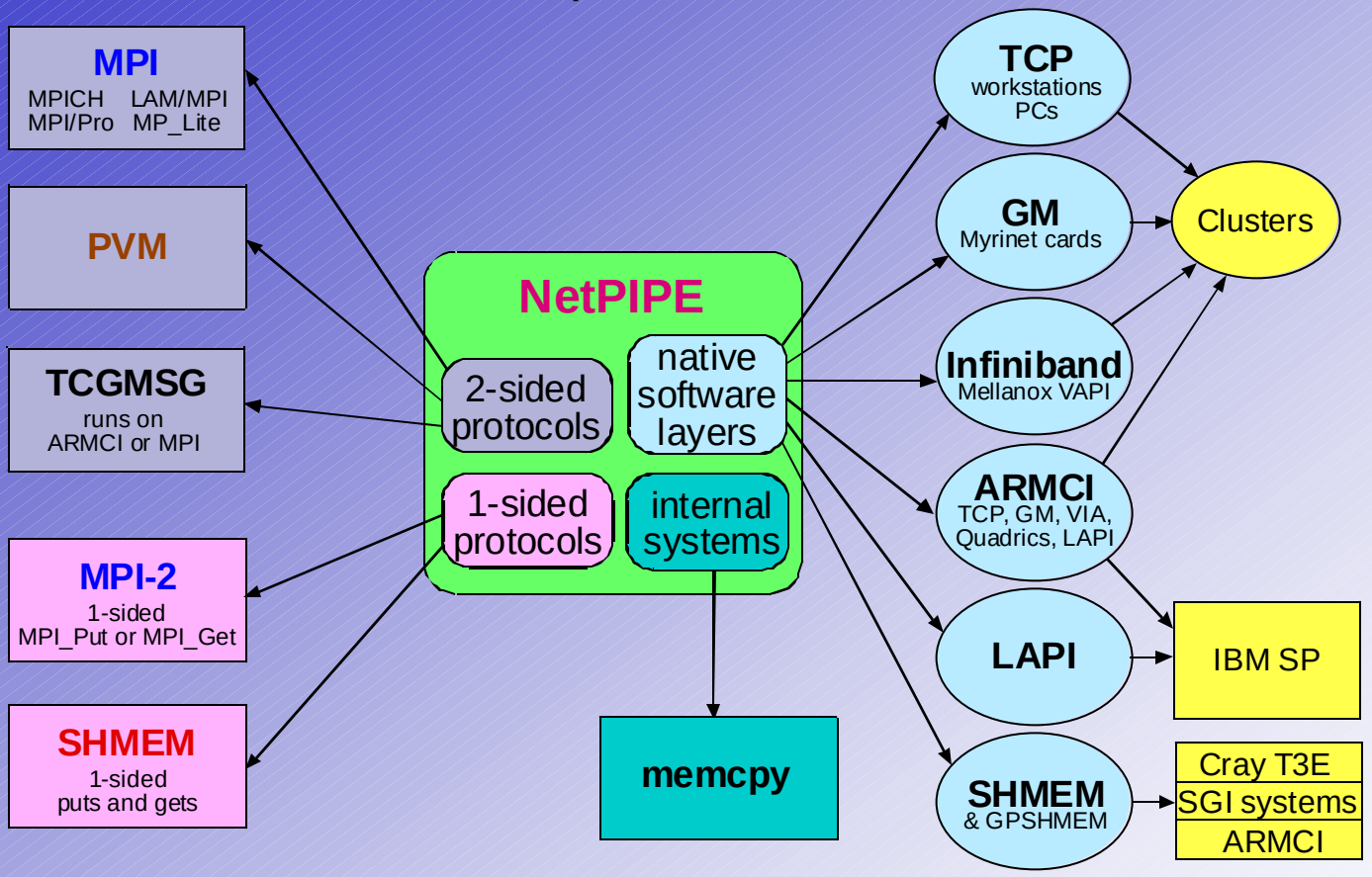
- Overlapping pair-wise ping-pong tests.
  - Must consider synchronization if not using bi-directional communications.



Line speed vs
end-point limited

- Investigate other methods for testing the global network.
  - Evaluate the full range from simultaneous nearest neighbor communications to all-to-all.

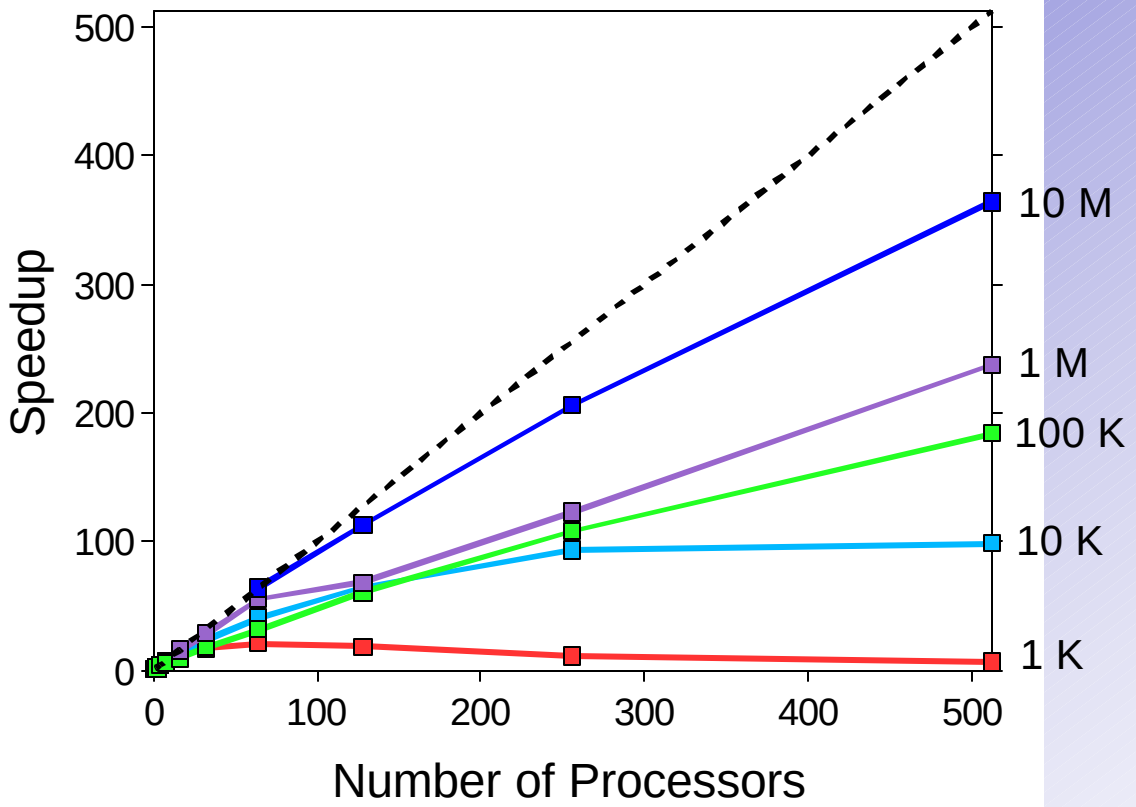
Network Protocol Independent Performance Evaluator



+ Basic send/recv with options to guarantee pre-posting or use MPI_ANY_SOURCE.
+ Option to measure performance without cache effects.
+ One-sided communications using either Get or Put, with or without fence calls.
+ Measure performance or do an integrity test.

http://www.scl.ameslab.gov/Projects/NetPIPE/

ALCMD Scaling on the Cray T3E



~75 MFlops / node
340 ==> 160 MB / sec max with a 20 us latency
3D toroid topology, but can't map to it

A 1,000 atom run for 20,000 time steps on 16 nodes takes 2.2 minutes
A 10,000 atom run for 20,000 time steps on 64 nodes takes 5.4 minutes
A 100,000 atom run for 20,000 time steps on 128 nodes takes 25 minutes
A 1,000,000 atom run for 20,000 time steps on 512 nodes takes 56 minutes

ALCMD – typical runs on the IBM SP

• 1,000 atoms	4 procs	20 hrs	10,000,000 steps
• 10,000 atoms	16 procs	6 hrs	1,000,000 steps
• 100,000 atoms	64 procs	15 hrs	1,000,000 steps
• 1,000,000 atoms	256 procs	4 hrs	100,000 steps
• 10,000,000 atoms	1024 procs	10 hrs	100,000 steps

Keep above 1000 atoms / processor.

Parallel efficiency kept above 75%.

110-150 MFlops / processor